# cribsheet

April 8, 2020

# 1 Exam Crib Sheet

## 1.1 Collections

### 1.1.1 `ArrayList`

- length can be changed dynamically
- index starts at zero; goes to length-1

**Imports** `import Java.util.ArrayList`

**Field Declaration** `private Arraylist<ElementType>;`

**Creation** `ArrayList<ElementType> listName = new Arraylist<>();`

**Methods**

- `ArrayList.clear()` → empty the list
- `ArrayList.add(Element)` → append the `Element` to the list
- `ArrayList.size()` → return the number of elements in the list
- `ArrayList.remove(int index)` → remove the element at index from the list
- `ArrayList.get(int index)` → return the element in the list at index
- `ArrayList.addAll(otherCollection)` → add an entire other collection object to `ArrayList`

### 1.1.2 `Array`

- fixed-size collection
- can store primitive types and references

**No Imports!**

- import `Java.util.Arrays` for useful features tho

**Field Declaration** `String[] shoebox;` → an array of strings

`public shoebox[] = {"words", "words"};` → no length needed; comes from initialized variables

`anArray = int[10]` → holds ten `int`s

`String[][]` → an array of arrays

**Access** `shoebox[1]` → array index from 0 to n-1

**Methods**

- Array methods:
  - `Array.length` → *NO PARENTHESES!* returns the length of the array
- Static methods from `Java.util.Arrays`:
  - `Arrays.asList(array);` → a List interface into `array`
  - `Arrays.equals( type array1[], type array2[] );` → returns true if `array1` and `array2` are equal
  - `Arrays.sort(arr);` → sort `arr` into ascending numerical order
  - `Arrays.binarySearch(arr[], key);` → find `key` in `arr[]` by bisection search. `arr[]` must be sorted.
  - `Arrays.fill(arr[], value);` → make every element in `arr` into `value`
- Other:
  - `System.ArrayCopy( source, sourcePos, dest, destPos, length );` → copy `length` elements from `source` to `dest`
  - will go like:
    * `source[sourcePos]`→`dest[destPos]`
    * `source[sourcePos + 1]`→`dest[destPos + 1]`
    * `...`
    * `source[sourcePos + length - 1]` → `dest[destPos + length - 1]`
    * elements in `dest` before `destPos` are not affected

### 1.1.3 `HashMap`

- a primitive database based on key/value mappings
- need to declare a key type and a value type
- unidirectional: you can look up a value with a key but not a key from a value

**Imports**

**Field Declaration**

**Creation** `Hashmap<keyType, valueType> hm = new HashMap<>();`

**Methods**

- `hm.put(Key, Value)` → add a new key/value pair to the map
- `hm.get(Key)` → return the value associated to `Key` in the map

[ ]:
```

```